EECE5644: Assignment #2

Machine Learning and Pattern Recognition

Name : Nikita Vinod Mandal

NUID : 002826995

Question 1:

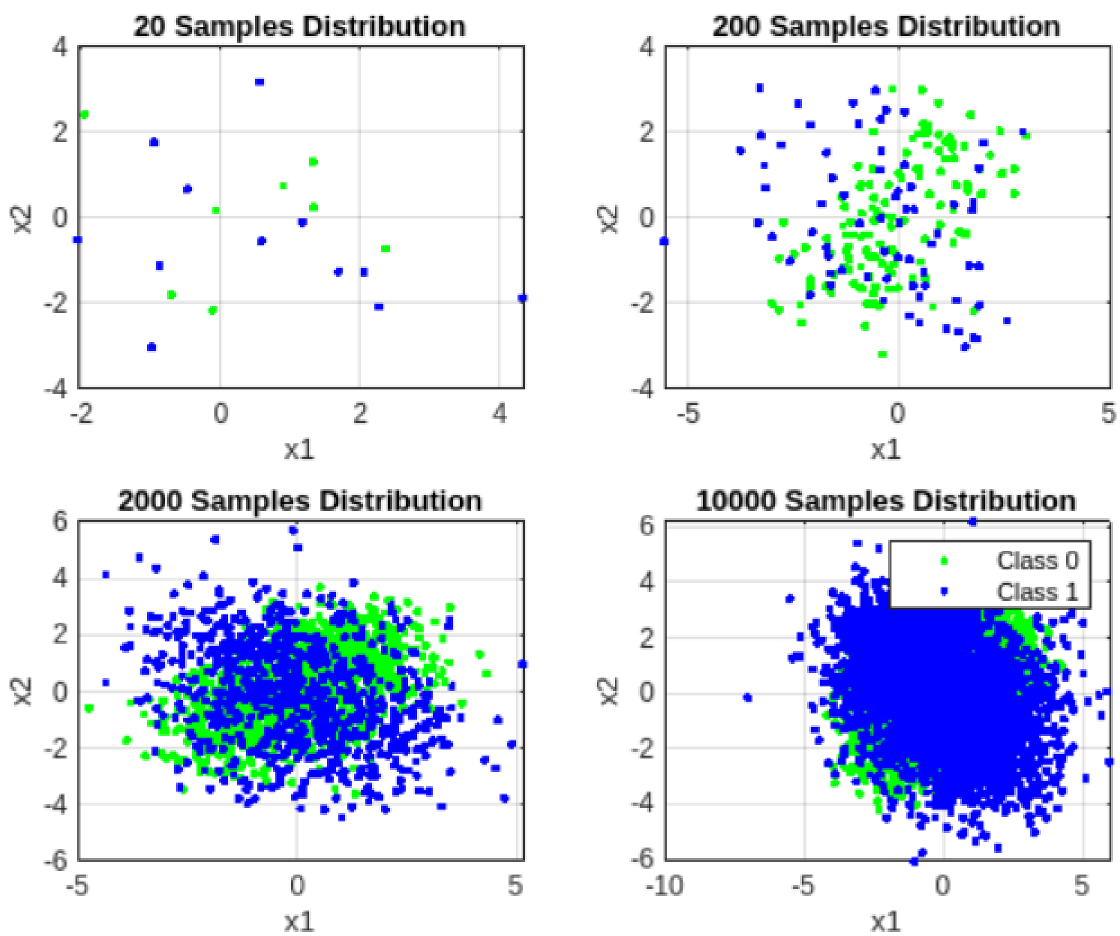


Fig. 1.1: Datasets

$$(D=1)=\frac{P(x|L1)}{P(x|L0)} \geq \frac{\lambda_{10}-\lambda_{00}}{\lambda_{01}-\lambda_{11}} * \frac{P(L0)}{P(L1)} = \gamma \ \ (D=0)$$

To minimize probability of misclassifications the cost for incorrect classification should be 1 and the cost for correct classifications should be 0 which results in the gamma shown below.

$$(D=1)=\frac{P(x|L1)}{P(x|L0)} \geq \frac{1-0}{1-0} * \frac{0.6}{0.4} = 1.86 = \gamma \ \ (D=0)$$

Plots of the ROC curve with the calculated ideal minimum error point as well as the minimum error point estimated from the generated validation data is shown in Figure 1.2. The probability of error versus Gamma with the calculated and estimated minimum error points marked are shown in Figure 1.3.

$$\frac{P(L=1|x)}{P(L=0|x)} \underset{D=0}{\overset{D=1}{\gtrless}} \frac{\lambda_{10}-\lambda_{00}}{\lambda_{01}-\lambda_{11}} \times \frac{P(L_0)}{P(L_1)} = r$$

$$\frac{P(L=1|x)}{P(L=0|x)} \underset{D=0}{\overset{D=1}{\gtrless}} \frac{1-0}{1-0} \times \frac{0.6}{0.4} = r = 1.5$$

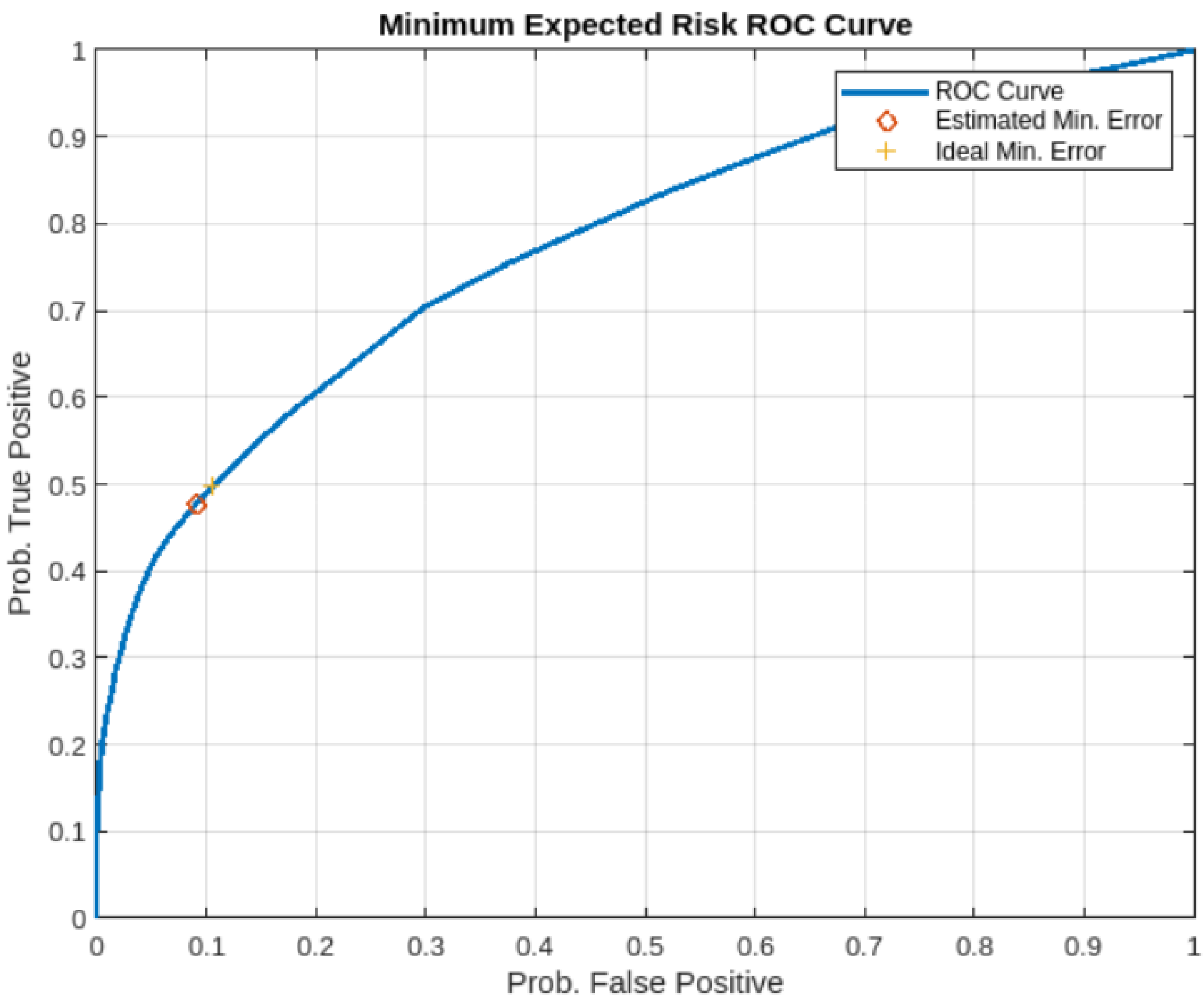


Fig 1.2: ROC Curve for known Ideal Classification Case

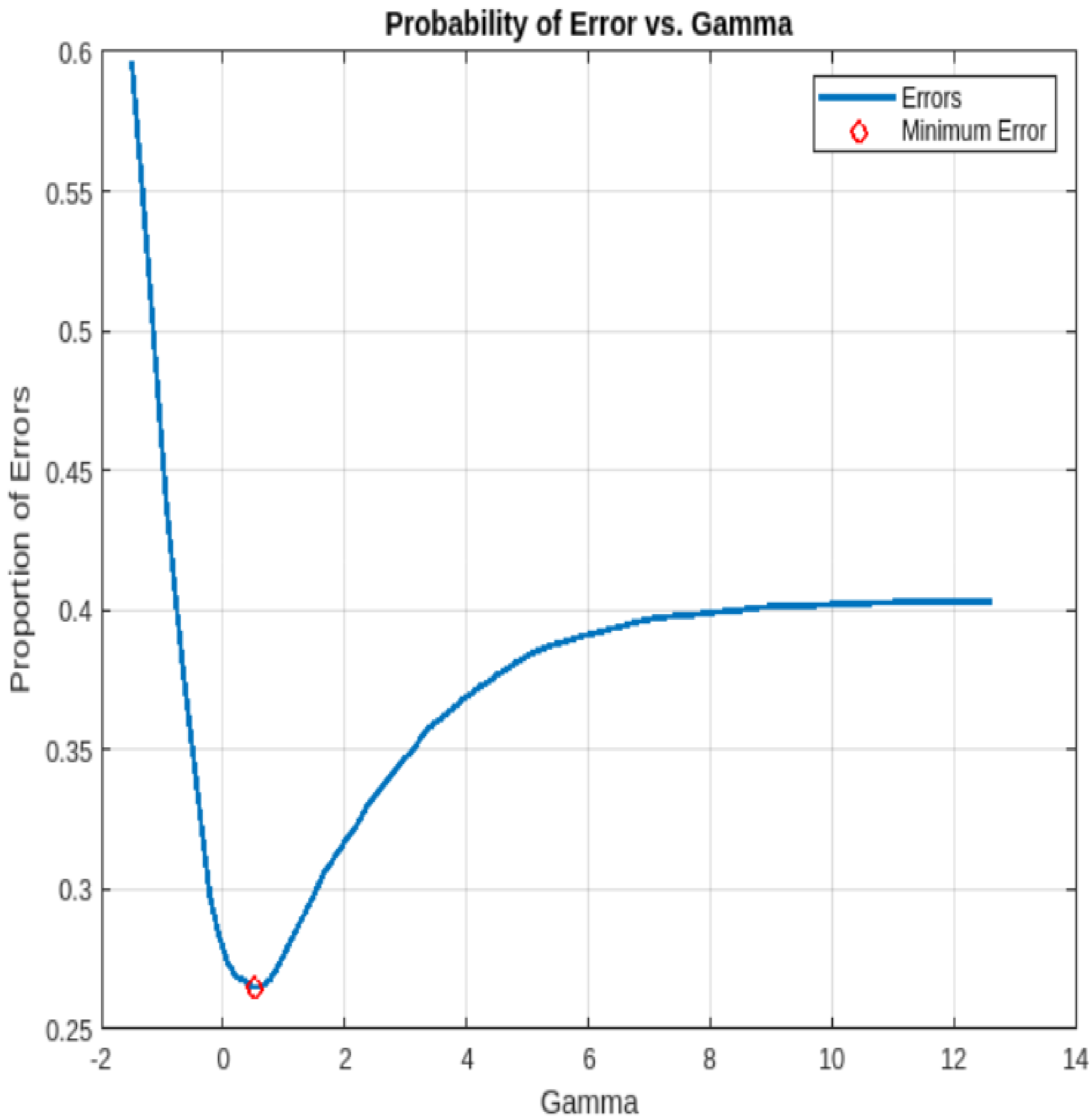


Fig 1.3: Probability of Error Curve for Ideal Classification

Figure 1.4 shows the decision space for each distribution along with equipluve contours of the discriminant function.

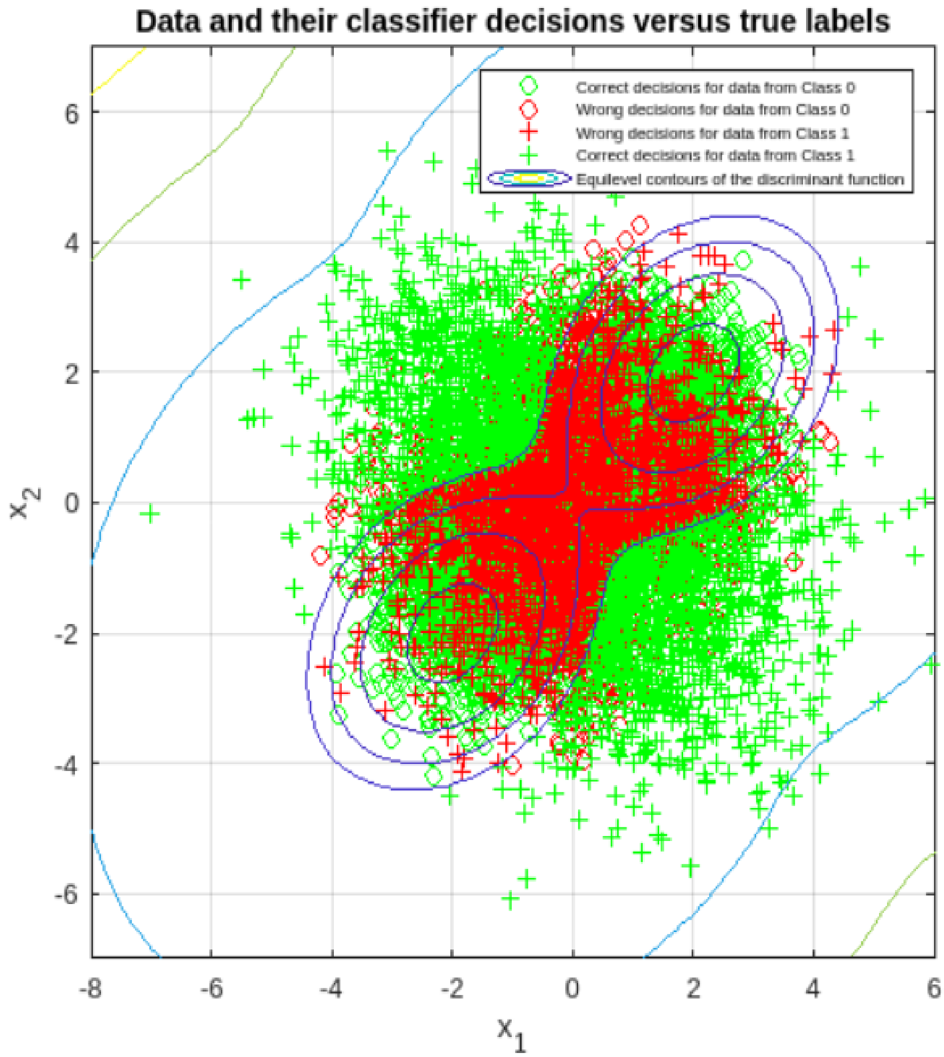


Fig 1.4: Decision Boundary of Ideal Classifier

For maximum likelihood parameter estimation techniques were used to train logistic linear and logistic quadratic based approximation of class label posterior functions given as a sample. This training was performed on each of the three separate training datasets consisting of 20, 200 and 2000 samples respectively and was then used to classify samples from the 10000 sample validation data set.

The logistic function is defined as follows:

$$h(x, w) = \frac{1}{1 + e^{w^T z(x)}}$$

For the linear logistic function $z(x) = [1\ x_1 x_2]^T$

For the quadratic logistic function $z(x) = [1\ x_1 x_2 x_1^2 x_1 * x_2 x_2^2]^T$

The **w** vectors are estimated using numerical optimization techniques with the cost function.

$\widehat{\theta_{ML}} = -\frac{1}{N}\sum_1^N l_n \ln\big(h(x_n, \theta)\big) + (1 - l_n) \ln\left(1 - h(x_n, \theta)\right)$

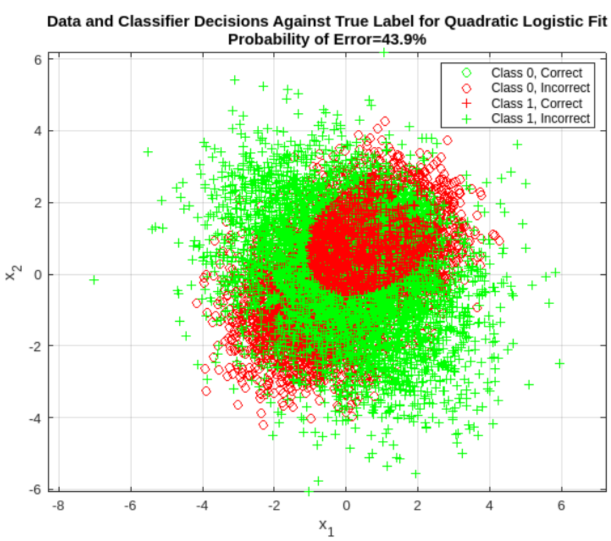
The minimum expected risk classification criteria are then.

$(l_n{=}1) \qquad \hat{w}^T z(x) \gtrless 0 \qquad (l_n{=}0)$

Table below contains a summary of the resulting probability of errors from classifying the 10000 sample validation data set using each of the 3 training data sets. The data shows that for both the linear and quadratic estimation functions the probabilities of error decrease as the number of points in the training datasets increase. Additionally, the quadratic logistic function significantly outperformed the linear logistic function in all cases.

Training Dataset	Linear	Quadratic
20	0.4920	0.3108
200	0.4348	0.2829
2000	0.4007	0.2741

Figures 1.5.1, 1.5.2, 1.6.1, 1.6.2, 1.7.1 and 1.7.2 show the data points of X with classifier decisions and true labels marked.



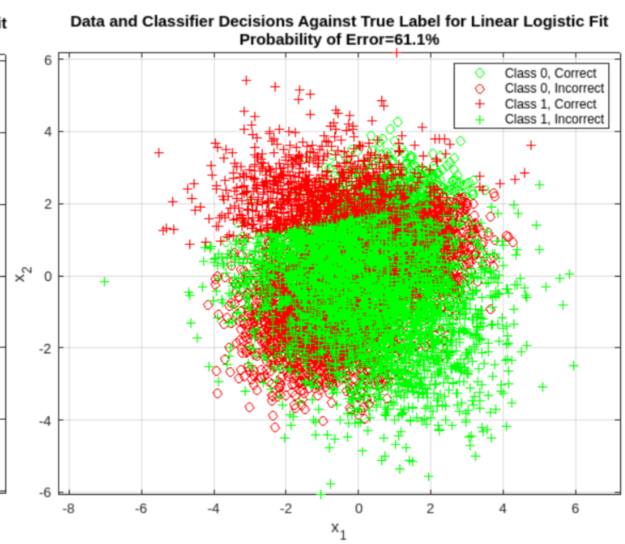
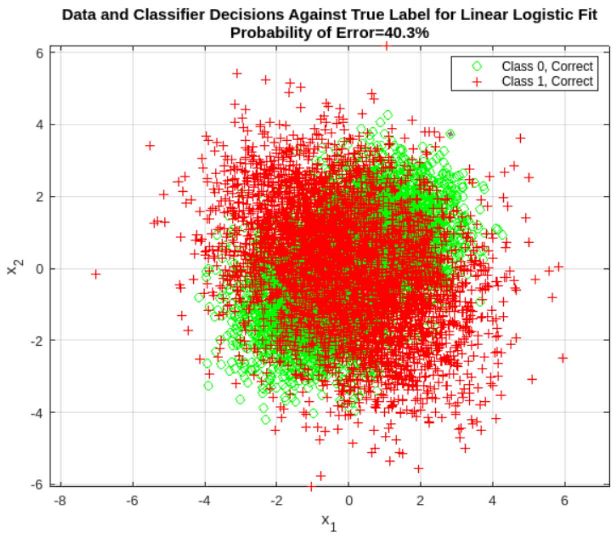


Fig 1.5.1: Classifier for Linear Logistic Fit on D20 training data **Fig 1.5.2:** Classifier for Quadratic Logistic Fit on D20 training data



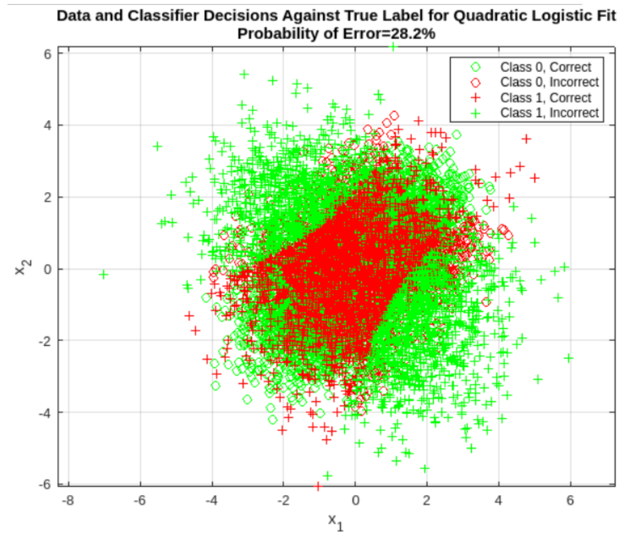
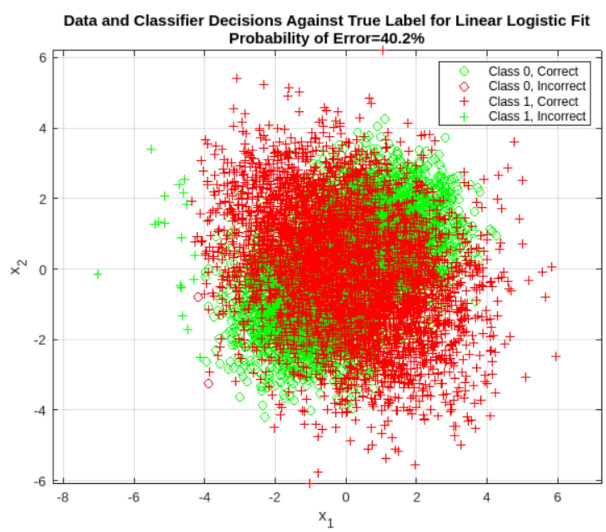


Fig 1.6.1: Classifier for Linear Logistic Fit on D200 training data **Fig 1.6.1:** Classifier for Quadratic Logistic Fit on D200 training data



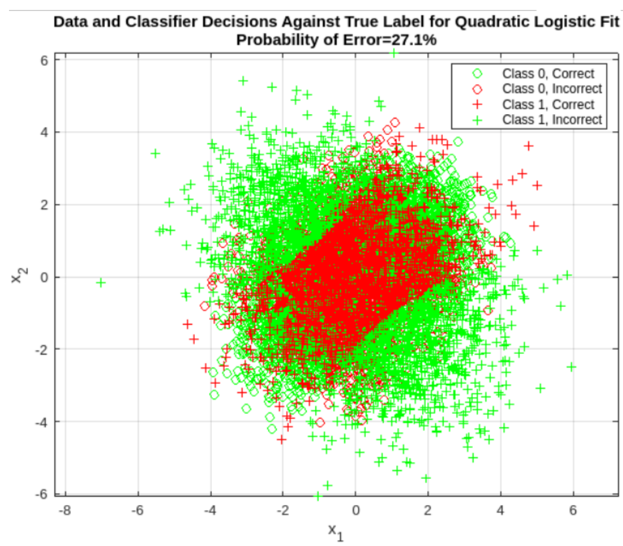


Fig 1.7.1: Classifier for Linear Logistic Fit on D2000 training data **Fig 1.7.1:** Classifier for Quadratic Logistic Fit on D2000 training data

Question 2:

Question 2 :-

- The data is generated from a Gaussian Mixture Model (GMM) with three components.
  Each component is characterized by a mean vector ('meanVectors'), a covariance matrix ('covMatrices') and a prior probabilities ('priors').

- The input data is transformed into a cubic polynomial feature space using the 'cubic_transformation' function.

- ML (Maximum Likelihood) Estimation :-
  In the linear regression model, we assume that the relationship between the input features (X) and the output (y) is given by
  $y = X \cdot \theta + \text{noise}$.
  where
  X is the matrix of input features
  θ is the vector of parameters to be estimated
  noise represents the random noise in the data.

The ML estimation aims to find the values of θ that maximizes the likelihood of observing the given data.

Assuming the noise follows a gaussian distribution the likelihood function is given by

$$L(\theta) = \prod_{i=0}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - x_i\theta)^2}{2\sigma^2}\right)$$

where,
N is the number of data points
y_i is the i^{th} ~~observati~~ observed output.
X_i is the i^{th} row of the feature matrix X.
σ^2 is the variance of the noise.

The Log likelihood function is often used for numerical stability.

$$\text{Log-Likelihood}(\theta) = -\frac{N}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - X_i\cdot\theta)^2$$

- Mean Squared Error (MSE)

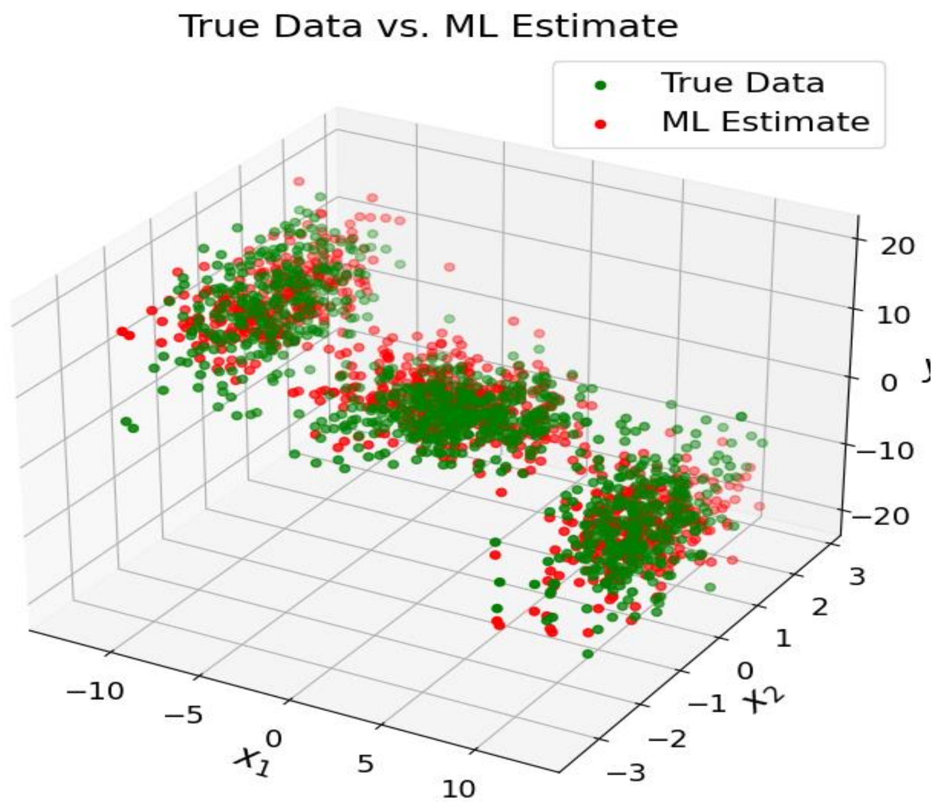
The mean squared error is a measure of the average squared difference between the predicted values and the true values. For a set of predictions $\hat{y}$ and true values y, the MSE is given by

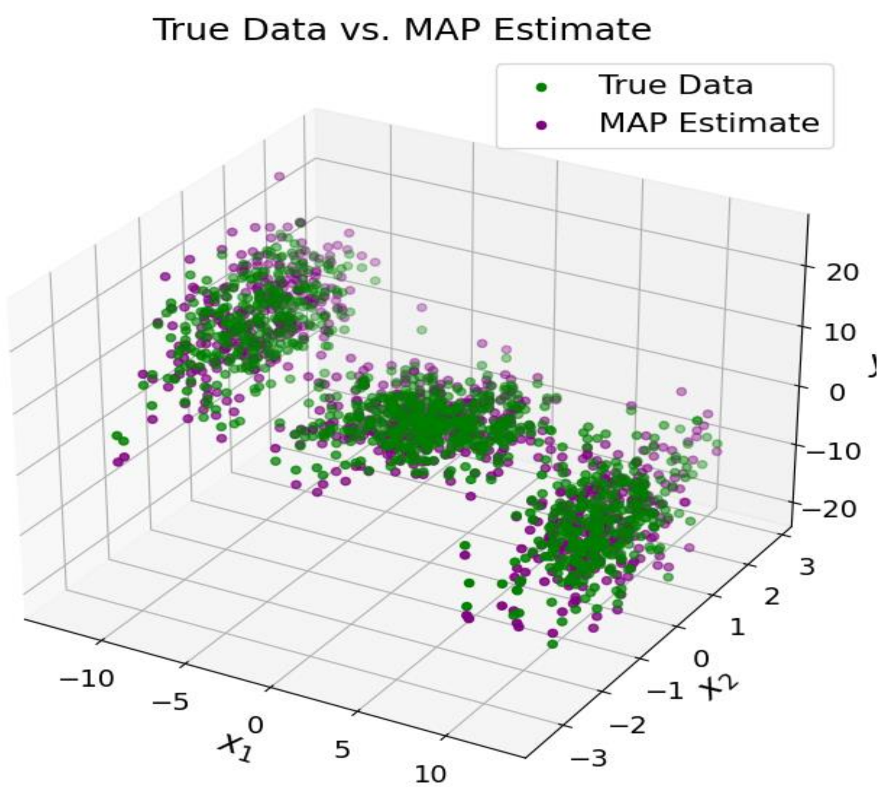
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

This is the objective function that is minimized during the training process to find the optimal values of θ, either through ML or MAP estimation.

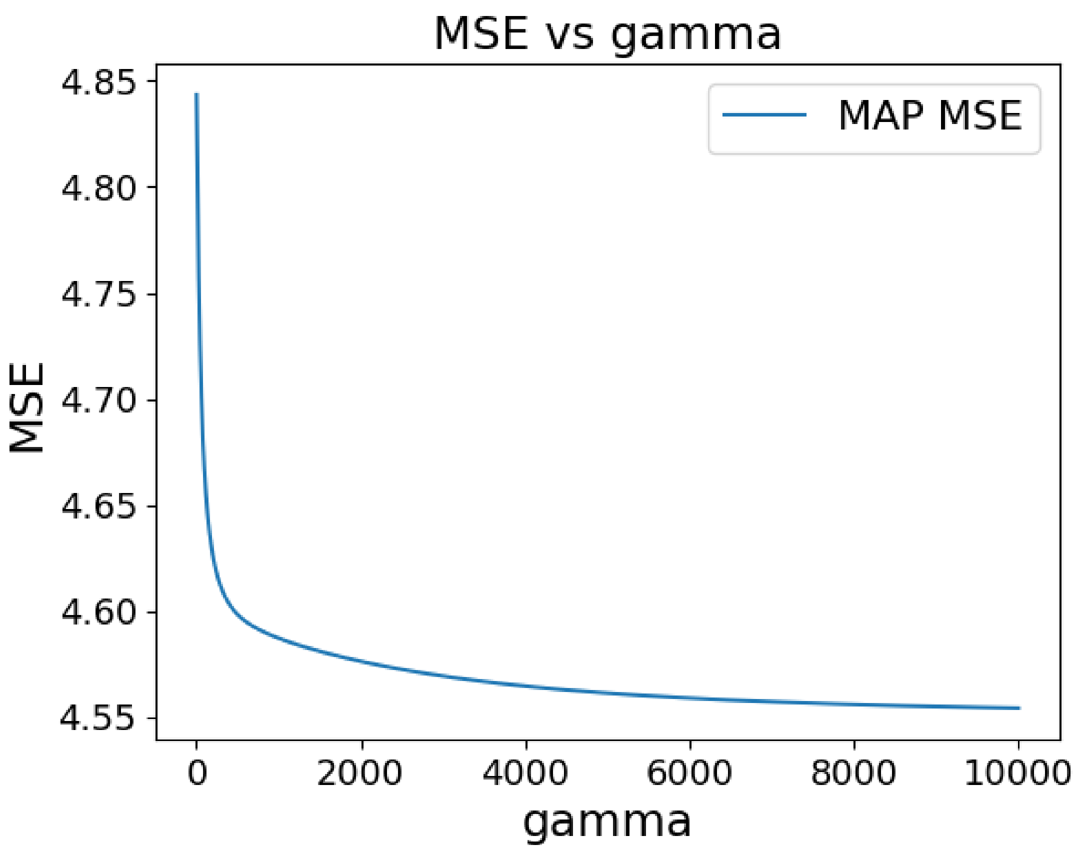
```
MLMAP,data.shape (3, 1000)
MLMAP,xValidate.shape,yValidate.shape: (2, 1000) (1000,)
xTrain, yTrain, xValidate, yValidate (2, 100) (100,) (2, 1000) (1000,)
xTrain, yTrain, xValidate, yValidate (100, 2) (100,) (1000, 2) (1000,)
10 batches of size 10:
theta start:
[ 1.38365602 -0.34458338 -0.50803804 -1.3134156  -0.07269325  0.21102434
  1.41292115  0.0759375  -0.11541498  0.08578218]
theta MLE:
[ 1.39596127 -0.36279139 -0.50864559 -0.01575003 -0.04368055  0.22082408
 -0.00592023 -0.00283205 -0.08601787  0.08746708]
theta MAP:
[[-0.04734302]
 [ 0.15571711]
 [ 0.13743308]
 [ 0.00581479]
 [-0.02772774]
 [-0.02206103]
 [-0.01173742]
 [-0.00290605]
 [-0.00571601]
 [ 0.17502425]]

MSE GD: 7.988058688778336
MSE MAP: 4.843545473454957
<Figure size 640x480 with 0 Axes>
<Figure size 640x480 with 0 Axes>
```


True Data vs. ML Estimate
True Data
ML Estimate
20
10
0
−10
−20
3
2
1
0
−1
−2
−3
x_2
−10
−5
0
5
10
x_1

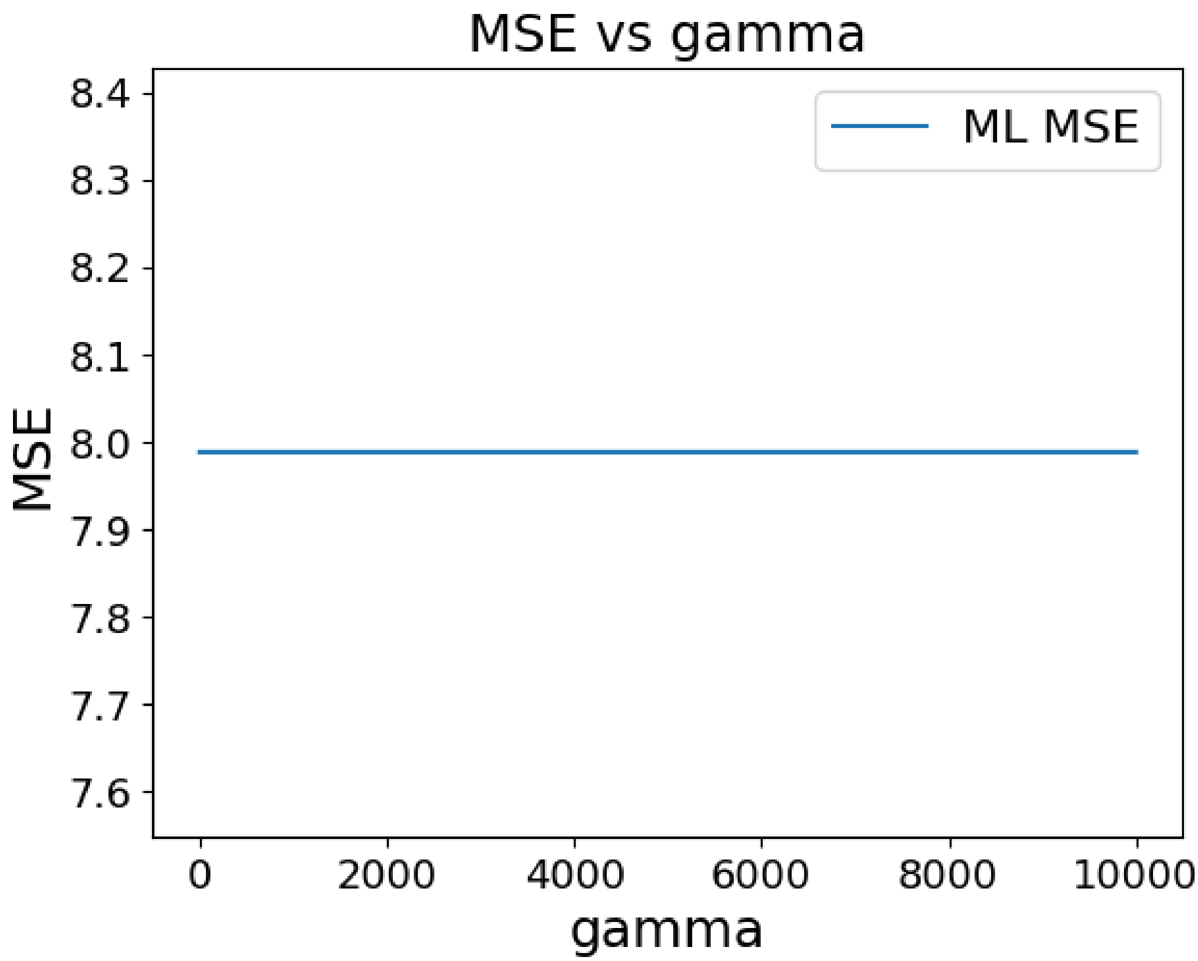

True Data vs. MAP Estimate
True Data
MAP Estimate
20
10
0
−10
−20
3
2
1
0
−1
−2
−3
x_2
−10
−5
0
5
10
x_1

10 batches of size 10:



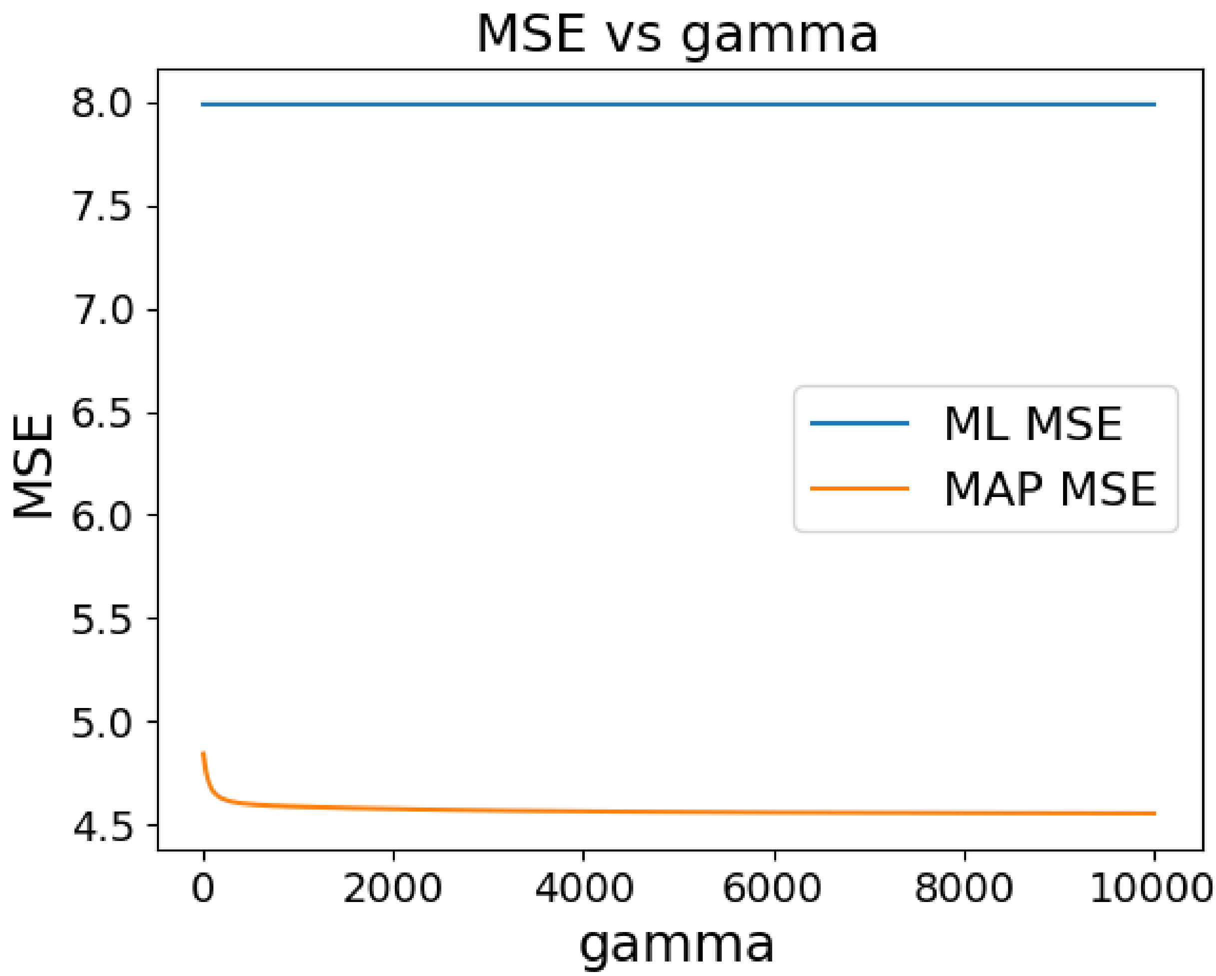
MSE vs Gamma (Regularization Parameter)

Line plot showing how Mean Squared Error (MSE) varies with the regularization parameter for the MAP estimate.



MSE vs Gamma for ML Estimate

Similar to the second plot, but for the Maximum Likelihood (ML) estimate.
Compares the performance of the ML estimate under different regularization settings.



Comparison of MSE for ML and MAP Estimates

Line plots showing how MSE varies with the regularization parameter for both ML and MAP estimates.

Question 3:

Question 3 :-

The objective is to find $[x, y]^T$ coordinate position with the ~~lowest~~ highest probability given the prior distribution as well as the range measurements from each of the K reference ~~to~~ coordinates.

$$\begin{bmatrix} x_{Map} \\ y_{map} \end{bmatrix} = \arg\max_{\begin{bmatrix} x \\ y \end{bmatrix}} p\left(\begin{bmatrix} x \\ y \end{bmatrix} \Big| \{r_1 \ldots r_K\} \right) \quad \ldots (1)$$

$$= \arg\max \left((2\pi \sigma_x \sigma_y)^{-1} e^{-\frac{1}{2}[x \;\; y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \right) \prod_{i=1}^{K} p\left(\begin{bmatrix} x \\ y \end{bmatrix} \Big| r_i \right) \quad \ldots (2)$$

$$= \arg\max_{\begin{bmatrix} x \\ y \end{bmatrix}} \ln\left((2\pi \sigma_x \sigma_y)^{-1} \right) + \ln\left(e^{-\frac{1}{2}[x \;\; y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \right) + \sum_{i=1}^{K} \ln p\left(\begin{bmatrix} x \\ y \end{bmatrix} \Big| r_i \right) \quad \ldots (3)$$

$$= \arg\max_{\begin{bmatrix} x \\ y \end{bmatrix}} \ln\left((2\pi \sigma_x \sigma_y)^{-1} \right) + \ln\left(e^{-\frac{1}{2}[x \;\; y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix}} \right) + \sum_{i=1}^{K} \ln p\left(\begin{bmatrix} x \\ y \end{bmatrix} \Big| r_i \right)$$

$$= \arg\max_{\begin{bmatrix} x \\ y \end{bmatrix}} -\frac{1}{2}[x \;\; y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln N(n_i | 0, \sigma_i^2) \quad \ldots (4)$$

$$= \arg\max -\frac{1}{2}[x \;\; y]\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi \sigma_i^2)^{-\frac{1}{2}} e^{-\frac{((r_i - d_i) - 0)^2}{2\sigma_i^2}} \right) \quad \ldots (5)$$

Date ______
Page ______

$$= \arg\max -\frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \ln\left((2\pi\sigma_i^2)^{-\frac{1}{2}}\right) + \ln\left(e^{\frac{-(r_i-d_i)^2}{2\sigma_i^2}}\right) \quad \ldots (6)$$

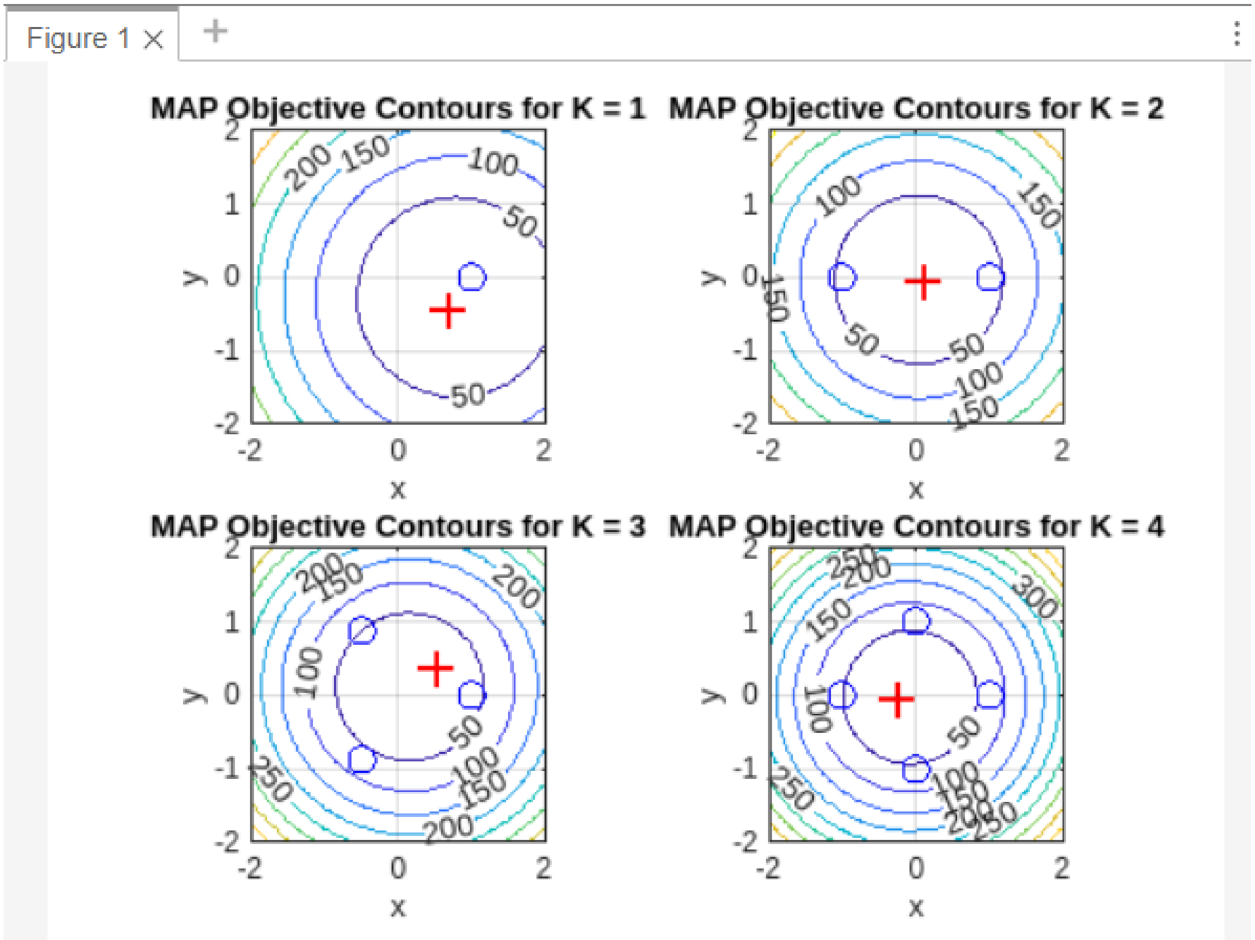
$$= \arg\max -\frac{1}{2}\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} -\frac{(r_i-d_i)^2}{2\sigma_i^2} \quad \ldots (7)$$

$$= \arg\min \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \frac{(r_i-d_i)^2}{\sigma_i^2} \quad \ldots (8)$$

$$= \arg\min \begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}^{-1}\begin{bmatrix} x \\ y \end{bmatrix} + \sum_{i=1}^{K} \frac{\left(r_i - \left\|\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix}\right\|\right)^2}{\sigma_i^2} \quad \ldots (9)$$

$$= \arg\min \left[\frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2}\right] + \sum_{i=1}^{K} \frac{\left(r_i - \left\|\begin{bmatrix} x \\ y \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix}\right\|\right)^2}{\sigma_i^2} \quad \ldots (10)$$

Using $\sigma_x = \sigma_y = 0.25$ and $\sigma_i = 0.3$.



A brief description of how the code works:

Initialization: The code initializes the random number generator, defines the standard deviations for the prior distribution, and creates a single figure to display subplots.

Loop Over K Values (Number of Landmarks): The code iterates over different values of K (number of landmarks). For each K, the following steps are performed:

- True Vehicle Location: The true location of the vehicle is set inside a circle with a unit radius centered at the origin.
- Landmark Positions: Landmark positions are evenly spaced on a circle with a unit radius centered at the origin.
- Measurement Generation: Range measurements are generated based on the specified model, incorporating measurement noise. If any measurement turns out to be negative, it is rejected and resampled.

- MAP Objective Computation: The MAP estimation objective function is computed on a grid. This objective function represents the negative log-likelihood of the vehicle position given the range measurements and prior knowledge.
- Contour Plotting: The code plots equilevel contours of the MAP estimation objective on a subplot. The true location of the vehicle and landmark positions are superimposed on the contour plot.

Analysis and Visualization:

- The contours of the MAP objective provide insights into the likelihood of different vehicle positions given the measurements and prior information.
- The true location of the vehicle is marked with a red cross ('+r'), and landmarks are marked with blue circles ('ob') in each subplot.
- The contours reveal regions of higher and lower likelihood, with the innermost contour representing a higher probability region.
- Visual assessment of the behavior of the MAP estimate relative to the true position can be done by observing the position of the innermost contour. If the MAP estimate is close to the true position, it suggests accurate localization.

Conclusion:

- As K (number of landmarks) increases, the localization tends to improve. This is because more landmarks provide additional information, reducing ambiguity in the vehicle's position.
- The MAP estimate appears to get closer to the true position as K increases, indicating improved localization accuracy.
- The contours justify these conclusions by visually representing the likelihood of different positions, and the true position aligning with the high-probability regions supports the accuracy of the MAP estimate.

Question 4:

Question 4 :-

→ Let,

$$\lambda(\alpha_i/w_j) = \begin{cases} 0 & i=j \quad i,j=1,\dots c \\ \lambda_r & i=c+1 \\ \lambda_s & \text{otherwise.} \end{cases}$$

- Bayes classifiers minimize conditional risks as follows (defined):

$$R(\alpha_i/x) = \sum_{J=1}^{c} \lambda(\alpha_i/w_j)\, P(w_j/x)$$

.... For $i=1,\dots,c$ i.e multiclass case.

Based on the $\lambda(\alpha_i/w_j)$ conditional loss definition given,

$$R(\alpha_i/x) = \sum_{\substack{J=1\\J\neq i}}^{c} \lambda_s \cdot P(w_j/x) \qquad \text{for } i=1,\dots,c$$

$$= \lambda_s \sum_{\substack{J=1\\J\neq i}}^{c} P(w_j/x)$$

$$R(\alpha_i/x) = \lambda_s \cdot [1 - P(w_i/x)] \qquad \text{for } i=1,\dots,c.$$

For $i=c+1$, we get

$$R(\alpha_{c+1}/x) = \lambda r.$$

- Minimum risk is achieved if we decide w_i if $R(\alpha_i/x) \leq R(\alpha_{c+1}/x)$

$$R(\alpha_i/x) \leq R(\alpha_{c+1}/x)$$

$$\lambda_s \cdot [1 - P(w_i/x)] \leq \lambda_r$$

$$\boxed{P(w_i/x) \geq 1 - \frac{\lambda_r}{\lambda_s}}$$

① If we have $\lambda_r = 0$, then we always reject.

$$P(L=i/x) \geq 1 - \frac{0}{\lambda_s}$$

i.e $P(L=i/x) \geq 1$

Since $\lambda_r = 0$, cost associated with reject class is 0. Hence, decision rule will always decide the reject class.

②. $\lambda_r > \lambda_s$ then we will never reject.

$$P(L=i/x) \geq 1 - \frac{\lambda_r}{\lambda_s}$$

If $\lambda_r > \lambda_s$, RHS is less than 0.

$$\therefore \quad P(L=i/x) \geq \underbrace{1 - \frac{\lambda_r}{\lambda_s}}_{<0}$$

i.e. $P(x) > 0$

∴ Decision rule will always choose some class i. It will never select reject class.

. This is intuitive as cost of reject class is high and we wish to minimize risk.

Question 5:

Question 5:-

→ To find the Maximum Likelihood (ML) estimator for θ, we need to maximize the likelihood function $L(\theta|D)$, where $D = \{z_1, \ldots, z_N\}$ is the given data set.

The samples $z_1, \ldots z_N$ are iid i.e independent and identically distributed. Hence, the likelihood function can be written as the product of individual probabilities.

$$L = (\theta / D) = \Pi [P(z_N / \theta)]$$

For the categorical distribution $P(z_N|\theta)$ is given by

$$P(z_N|\theta) = \Pi [\theta_K \hat{}\, z_K] \quad \text{i.e } \Pi [\theta_K^{z_k}]$$

∴ The likelihood function becomes

$$L(\theta|D) = \Pi [\Pi [\theta_K^{z_K}]]$$

Taking the logarithm of the likelihood function,

$$\log(L(\theta|D)) = \Sigma [\Sigma [z_K * \log(\theta_K)]]$$

Now, to find the ML estimator of θ, we need to maximize the function w.r.t. θ.
However, the constraint is that the probabilities should sum up to 1.
i.e $\Sigma [\theta_k] = 1$.

To handle this constraint, we can introduce Lagrange multiplier and maximize the function,

$$\log(L(\theta|D)) + \lambda * (\Sigma[\theta_k] - 1).$$

Taking derivative w.r.t θ_k and setting it to 0, the ML estimator for θ,

$$\frac{\partial(\log(L(\theta|D)) + \lambda * (\Sigma[\theta_k] - 1))}{(\partial \theta_k)} = 0$$

$\therefore$ The ML estimator:

$$\theta_k = (\Sigma[z_k]) / N$$

where N is the total number of samples.

→ Lets consider the MAP estimator for θ, assuming a Dirichlet prior distribution with parameter α.

The MAP estimator maximizes the posterior probability, which is proportional to the product of the likelihood and the prior.

$$P(\theta|D) = P(D|\theta) * P(\theta)$$

Using Baye's Theorem,

$$P(\theta|D) = P(D|\theta) * P(\theta)$$
$$\propto L(\theta|D) * P(\theta)$$

Taking logarithm,

$\log(P(\theta|D)) \propto \log(L(\theta|D)) + \log(P(\theta))$

The Dirichlet prior distribution can be represented as,

$P(\theta) = \cancel{\frac{1}{B(\alpha)} *}\ 1/B(\alpha) * \Pi[\theta k \hat{} (\alpha k - 1)]$

where $B(\alpha)$ is the normalization constant.

Therefore, the MAP estimator for θ can be found by maximizing the following function:

$$\log(L(\theta|D)) + \log(P(\theta)) \propto \sum_{n=1}^{N}\left[\sum_{k=1}^{K}[z_{nk} * \log(\theta k)]\right] + \sum_{k=1}^{K}[(\alpha k-1) * \log(\theta k)] + \log(B(\alpha))$$

The MAP estimator can be obtained in a similar way as the ML estimator by taking derivatives and solving for θk, considering the probabilities should sum up to 1.

$\therefore$ Taking derivative w.r.to θk & setting it to 0.

$$\frac{\partial(\log(L(\theta|D)) + \log(P(\theta)))}{\partial\theta k} = 0$$

$\therefore$ The MAP estimator:

$$\theta k = \frac{\Sigma_{n=1}^{N}(z_{nk} + \alpha_k - 1)}{\Sigma_{k=1}^{K}\Sigma_{n=1}^{N}(z_{nk} + \alpha_k - 1)}$$

where N is the total number of samples, K is the number of categories and α_k is the parameter of the Dirichlet prior of category k.

Appendix:

Question 1:

```
clear all;
close all;

Part1 = 1;
Part2 = 1;
dimension = 2;

D.d100.N = 20;
D.d1000.N = 200;
D.d10k.N = 2000;
D.d20k.N = 10000;
DType = fieldnames(D);

p = [0.6, 0.4];

% Label 0 GMM
mu0 = [-1, -1; 1, 1]';
Sigma0(:,:,1) = [1, 0; 0, 1];
Sigma0(:,:,2) = [1, 0; 0, 1];
alpha0 = [0.5, 0.5];

% Label 1
mu1 = [-1, 1; 1, -1]';
Sigma1(:,:,1) = [2, 0; 0, 2];
Sigma1(:,:,2) = [2, 0; 0, 2];
alpha1 = [0.5, 0.5];

figure;

for ind = 1:length(DType)
    % Generate random labels based on class priors
    D.(DType{ind}).labels = rand(1, D.(DType{ind}).N) >= p(1);

    % Calculate class proportions
    D.(DType{ind}).N0 = sum(~D.(DType{ind}).labels);
    D.(DType{ind}).N1 = sum(D.(DType{ind}).labels);
    D.(DType{ind}).phat = [D.(DType{ind}).N0, D.(DType{ind}).N1] / D.(DType{ind}).N;

    % Generate samples for each class
    D.(DType{ind}).x = zeros(dimension, D.(DType{ind}).N);
    [D.(DType{ind}).x(:, ~D.(DType{ind}).labels), ...
     D.(DType{ind}).dist(:, ~D.(DType{ind}).labels)] = ...
        randGMM(D.(DType{ind}).N0, alpha0, mu0, Sigma0);

    [D.(DType{ind}).x(:, D.(DType{ind}).labels), ...
     D.(DType{ind}).dist(:, D.(DType{ind}).labels)] = ...
        randGMM(D.(DType{ind}).N1, alpha1, mu1, Sigma1);

    % Plot samples with different colors
```

```matlab
    subplot(2, 2, ind);
    plot(D.(DType{ind}).x(1, ~D.(DType{ind}).labels), ...
        D.(DType{ind}).x(2, ~D.(DType{ind}).labels), 'g.', 'DisplayName', 'Class 0');
    hold on;
    plot(D.(DType{ind}).x(1, D.(DType{ind}).labels), ...
        D.(DType{ind}).x(2, D.(DType{ind}).labels), 'b.', 'DisplayName', 'Class 1');
    grid on;
    xlabel('x1');
    ylabel('x2');
    title([num2str(D.(DType{ind}).N) ' Samples Distribution']);
end

legend('show');

% Part 1
if Part1
    % Evaluate class-conditional densities
    px0 = evalGMM(D.d20k.x, alpha0, mu0, Sigma0);
    px1 = evalGMM(D.d20k.x, alpha1, mu1, Sigma1);

    % Calculate discriminant scores
    discScore = log(px1 ./ px0);

    % Sort the discriminant scores
    sortDS = sort(discScore);

    % Define a range of threshold values
    logGamma = linspace(min(discScore) - eps, max(discScore) + eps, 100);

    % Calculate probabilities for each threshold
    prob = CalcProb(discScore, logGamma, D.d20k.labels, D.d20k.N0, D.d20k.N1, D.d20k.phat);

    % Define the ideal threshold based on the class priors
    logGamma_ideal = log(p(1) / p(2));

    % Make decisions based on the ideal threshold
    decision_ideal = discScore > logGamma_ideal;

    % Calculate error rates for the ideal threshold
    p10_ideal = sum(decision_ideal == 1 & D.d20k.labels == 0) / D.d20k.N0;
    p11_ideal = sum(decision_ideal == 1 & D.d20k.labels == 1) / D.d20k.N1;
    pFE_ideal = (p10_ideal * D.d20k.N0 + (1 - p11_ideal) * D.d20k.N1) / (D.d20k.N0 + D.d20k.N1);

    % Find the minimum probability of error
    [prob.min_pFE, prob.min_pFE_ind] = min(prob.pFE);

    % If there are multiple minima, choose the one closest to the ideal threshold
    if length(prob.min_pFE_ind) > 1
        [~, minDistTheory_ind] = min(abs(logGamma(prob.min_pFE_ind) - logGamma_ideal));
        prob.min_pFE_ind = prob.min_pFE_ind(minDistTheory_ind);
    end

    % Extract values for the minimum probability of error
```

```matlab
    minGAMMA = exp(logGamma(prob.min_pFE_ind));
    prob.min_FP = prob.p10(prob.min_pFE_ind);
    prob.min_TP = prob.p11(prob.min_pFE_ind);

    % Plot ROC curve and related information
    plotROC(prob.p10, prob.p11, prob.min_FP, prob.min_TP);
    hold all;
    plot(p10_ideal, p11_ideal, '+', 'DisplayName', 'Ideal Min. Error');
    plotMinPFE(logGamma, prob.pFE, prob.min_pFE_ind);
    plotDecisions(D.d20k.x, D.d20k.labels, decision_ideal);

    % Plot ERM contours
    plotERMContours(D.d20k.x, alpha0, mu0, Sigma0, alpha1, mu1, Sigma1, logGamma_ideal);
end

% Part 2: Classification with Maximum Likelihood Parameter Estimation

options = optimset('MaxFunEvals', 60000, 'MaxIter', 20000);

for ind = 1:length(DType)
    lin.x = [ones(1, D.(DType{ind}).N); D.(DType{ind}).x];
    lin.init = zeros(dimension + 1, 1);

    lin.theta = fminsearch(@(theta)(costFun(theta, lin.x, D.(DType{ind}).labels)), lin.init, options);
    lin.discScore = lin.theta' * [ones(1, D.d20k.N); D.d20k.x];
    gamma = 0;
    lin.prob = CalcProb(lin.discScore, gamma, D.d20k.labels, D.d20k.N0, D.d20k.N1, D.d20k.phat);

    quad.x = [ones(1, D.(DType{ind}).N); D.(DType{ind}).x;...
            D.(DType{ind}).x(1, :).^2;...
            D.(DType{ind}).x(1, :).*D.(DType{ind}).x(2, :);...
            D.(DType{ind}).x(2, :).^2];
    quad.init = zeros(2*(dimension + 1), 1);

    quad.theta = fminsearch(@(theta)(costFun(theta, quad.x, D.(DType{ind}).labels)), quad.init, options);
    quad.xScore = [ones(1, D.d20k.N); D.d20k.x; D.d20k.x(1, :).^2;...
              D.d20k.x(1, :).*D.d20k.x(2, :); D.d20k.x(2, :).^2];
    quad.discScore = quad.theta' * quad.xScore;
    gamma = 0;
    quad.prob = CalcProb(quad.discScore, gamma, D.d20k.labels, D.d20k.N0, D.d20k.N1, D.d20k.phat);

    plotDecisions(D.d20k.x, D.d20k.labels, lin.prob.decisions);
    title(sprintf('Data and Classifier Decisions Against True Label for Linear Logistic Fit\nProbability of
Error=%1.1f%%', 100*lin.prob.pFE));

    plotDecisions(D.d20k.x, D.d20k.labels, quad.prob.decisions);
    title(sprintf('Data and Classifier Decisions Against True Label for Quadratic Logistic Fit\nProbability of
Error=%1.1f%%', 100*quad.prob.pFE));
end

% Function Definitions

function [x, labels] = randGMM(N, alpha, mu, Sigma)
```

```
    d = size(mu, 1);
    cum_alpha = [0, cumsum(alpha)];
    u = rand(1, N);
    x = zeros(d, N);
    labels = zeros(1, N);

    for m = 1:length(alpha)
        ind = find(cum_alpha(m) < u & u <= cum_alpha(m + 1));
        x(:, ind) = randGaussian(length(ind), mu(:, m), Sigma(:, :, m));
        labels(ind) = m - 1;
    end
end

function x = randGaussian(N, mu, Sigma)
    n = length(mu);
    z = randn(n, N);
    A = Sigma^(1/2);
    x = A * z + repmat(mu, 1, N);
end

function cost = costFun(theta, x, labels)
    h = 1./(1 + exp(-x' * theta));
    cost = -1/length(h) * sum((labels' .* log(h) + (1 - labels)' .* (log(1 - h))));
end

function gmm = evalGMM(x, alpha, mu, Sigma)
    gmm = zeros(1, size(x, 2));

    for m = 1:length(alpha)
        gmm = gmm + alpha(m) * evalGaussian(x, mu(:, m), Sigma(:, :, m));
    end
end

function g = evalGaussian(x, mu, Sigma)
    [n, N] = size(x);
    invSigma = inv(Sigma);
    C = (2*pi)^(-n/2) * det(invSigma)^(1/2);
    E = -0.5 * sum((x - repmat(mu, 1, N)) .* (invSigma * (x - repmat(mu, 1, N))), 1);
    g = C * exp(E);
end

function prob = CalcProb(discScore, logGamma, labels, N0, N1, phat)
    for ind = 1:length(logGamma)
        prob.decisions = discScore >= logGamma(ind);
        Num_pos(ind) = sum(prob.decisions);
        prob.p10(ind) = sum(prob.decisions == 1 & labels == 0) / N0;
        prob.p11(ind) = sum(prob.decisions == 1 & labels == 1) / N1;
        prob.p01(ind) = sum(prob.decisions == 0 & labels == 1) / N1;
        prob.p00(ind) = sum(prob.decisions == 0 & labels == 0) / N0;
        prob.pFE(ind) = prob.p10(ind) * phat(1) + prob.p01(ind) * phat(2);
    end
end
```

```
function plotContours(x, alpha, mu, Sigma)
  figure

  if size(x, 1) == 2
    plot(x(1, :), x(2, :), 'b.');
    xlabel('x_1');
    ylabel('x_2');
    title('Data and Estimated GMM Contours');
    axis equal;
    hold on;
    rangex1 = [min(x(1, :)), max(x(1, :))];
    rangex2 = [min(x(2, :)), max(x(2, :))];
    [x1Grid, x2Grid, zGMM] = contourGMM(alpha, mu, Sigma, rangex1, rangex2);
    contour(x1Grid, x2Grid, zGMM);
    axis equal;
  end
end

function [x1Grid, x2Grid, zGMM] = contourGMM(alpha, mu, Sigma, rangex1, rangex2)
  x1Grid = linspace(floor(rangex1(1)), ceil(rangex1(2)), 101);
  x2Grid = linspace(floor(rangex2(1)), ceil(rangex2(2)), 91);
  [h, v] = meshgrid(x1Grid, x2Grid);
  GMM = evalGMM([h(:)';v(:)'], alpha, mu, Sigma);
  zGMM = reshape(GMM, 91, 101);
end

function plotROC(p10, p11, min_FP, min_TP)
  figure;
  plot(p10, p11, 'DisplayName', 'ROC Curve', 'LineWidth', 2);
  hold on;
  plot(min_FP, min_TP, 'o', 'DisplayName', 'Estimated Min. Error', 'LineWidth', 2);
  xlabel('Prob. False Positive');
  ylabel('Prob. True Positive');
  title('Minimum Expected Risk ROC Curve');
  legend('show');
  grid on;
  box on;
end

function plotMinPFE(logGamma, pFE, min_pFE_ind)
  figure;
  plot(logGamma, pFE, 'DisplayName', 'Errors', 'LineWidth', 2);
  hold on;
  plot(logGamma(min_pFE_ind), pFE(min_pFE_ind), 'ro', 'DisplayName', 'Minimum Error', 'LineWidth', 2);
  xlabel('Gamma');
  ylabel('Proportion of Errors');
  title('Probability of Error vs. Gamma');
  grid on;
  legend('show');
end

function plotDecisions(x, labels, decisions)
  ind00 = find(decisions == 0 & labels == 0);
```

```
    ind10 = find(decisions == 1 & labels == 0);
    ind01 = find(decisions == 0 & labels == 1);
    ind11 = find(decisions == 1 & labels == 1);
    figure;
    plot(x(1, ind00), x(2, ind00), 'og', 'DisplayName', 'Class 0, Correct');
    hold on;
    plot(x(1, ind10), x(2, ind10), 'or', 'DisplayName', 'Class 0, Incorrect');
    hold on;
    plot(x(1, ind01), x(2, ind01), '+r', 'DisplayName', 'Class 1, Correct');
    hold on;
    plot(x(1, ind11), x(2, ind11), '+g', 'DisplayName', 'Class 1, Incorrect');
    hold on;
    axis equal;
    grid on;
    title('Data and their classifier decisions versus true labels');
    xlabel('x_1');
    ylabel('x_2');
    legend('AutoUpdate', 'off');
    legend('show');
end

function plotERMContours(x, alpha0, mu0, Sigma0, alpha1, mu1, Sigma1, logGamma_ideal)
    horizontalGrid = linspace(floor(min(x(1, :))), ceil(max(x(1, :))), 101);
    verticalGrid = linspace(floor(min(x(2, :))), ceil(max(x(2, :))), 91);
    [h, v] = meshgrid(horizontalGrid, verticalGrid);
    discriminantScoreGridValues = log(evalGMM([h(:)'; v(:)'], alpha1, mu1, Sigma1)) - log(evalGMM([h(:)'; v(:)'],
alpha0, mu0, Sigma0)) - logGamma_ideal;
    minDSGV = min(discriminantScoreGridValues);
    maxDSGV = max(discriminantScoreGridValues);
    discriminantScoreGrid = reshape(discriminantScoreGridValues, 91, 101);
    contour(horizontalGrid, verticalGrid, discriminantScoreGrid, [minDSGV * [0.9, 0.6, 0.3], 0, [0.3, 0.6, 0.9] *
maxDSGV]);
    lgd=legend('Correct decisions for data from Class 0', 'Wrong decisions for data from Class 0', 'Wrong decisions
for data from Class 1', 'Correct decisions for data from Class 1', 'Equilevel contours of the discriminant function');

    set(lgd, 'FontSize', 6);
end
```

Question 2:

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import multivariate_normal
from sklearn.metrics import confusion_matrix
from math import ceil, floor
from sklearn.preprocessing import PolynomialFeatures
import numpy as np
from matplotlib import pyplot
import pylab
from mpl_toolkits.mplot3d import Axes3D

def MLMAP():
    Ntrain = 100
    data = generateData(Ntrain)
    plot3(data[0, :], data[1, :], data[2, :])
    xTrain = data[0:2, :]
    yTrain = data[2, :]

    Ntrain = 1000
    data = generateData(Ntrain)
    plot3(data[0, :], data[1, :], data[2, :])
    print("MLMAP,data.shape", data.shape)
    xValidate = data[0:2, :]
    yValidate = data[2, :]
    print("MLMAP,xValidate.shape,yValidate.shape:", xValidate.shape, yValidate.shape)

    return xTrain, yTrain, xValidate, yValidate

def generateData(N):
    gmmParameters = {}
    gmmParameters['priors'] = [.3, .4, .3]  # priors should be a row vector
    gmmParameters['meanVectors'] = np.array([[-10, 0, 10], [0, 0, 0], [10, 0, -10]])
    gmmParameters['covMatrices'] = np.zeros((3, 3, 3))
    gmmParameters['covMatrices'][:, :, 0] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    gmmParameters['covMatrices'][:, :, 1] = np.array([[8, 0, 0], [0, .5, 0], [0, 0, .5]])
    gmmParameters['covMatrices'][:, :, 2] = np.array([[1, 0, -3], [0, 1, 0], [-3, 0, 15]])
    x, labels = generateDataFromGMM(N, gmmParameters)
    return x

def generateDataFromGMM(N, gmmParameters):
    priors = gmmParameters['priors']  # priors should be a row vector
    meanVectors = gmmParameters['meanVectors']
    covMatrices = gmmParameters['covMatrices']
```

```python
    n = meanVectors.shape[0]  # Data dimensionality
    C = len(priors)  # Number of components
    x = np.zeros((n, N))
    labels = np.zeros((1, N))
    u = np.random.random((1, N))
    thresholds = np.zeros((1, C + 1))
    thresholds[:, 0:C] = np.cumsum(priors)
    thresholds[:, C] = 1
    for l in range(C):
        indl = np.where(u <= float(thresholds[:, l]))
        Nl = len(indl[1])
        labels[indl] = (l + 1) * 1
        u[indl] = 1.1
        x[:, indl[1]] = np.transpose(np.random.multivariate_normal(meanVectors[:, l], covMatrices[:, :, l], Nl))

    return x, labels

def plot3(a, b, c, mark="o", col="r"):
    pylab.ion()
    fig = pylab.figure()
    ax = Axes3D(fig)
    ax.scatter(a, b, c, marker=mark, color=col)
    ax.set_xlabel("x1")
    ax.set_ylabel("x2")
    ax.set_zlabel("y")
    ax.set_title('Training Dataset')

np.set_printoptions(suppress=True)
np.random.seed(11)
plt.rc('font', size=18)
plt.rc('axes', titlesize=18)
plt.rc('axes', labelsize=18)
plt.rc('xtick', labelsize=14)
plt.rc('ytick', labelsize=14)
plt.rc('legend', fontsize=16)
plt.rc('figure', titlesize=18)

def batchify(X, y, batch_size, N):
    X_batch = []
    y_batch = []
    for i in range(0, N, batch_size):
        nxt = min(i + batch_size, N + 1)
        X_batch.append(X[i:nxt, :])
        y_batch.append(y[i:nxt])
    return X_batch, y_batch
```

```python
def gradient_descent(loss_func, theta0, X, y, N, *args, **kwargs):
    max_epoch = kwargs['max_epoch'] if 'max_epoch' in kwargs else 200
    alpha = kwargs['alpha'] if 'alpha' in kwargs else 0.1
    epsilon = kwargs['tolerance'] if 'tolerance' in kwargs else 1e-6
    batch_size = kwargs['batch_size'] if 'batch_size' in kwargs else 10

    X_batch, y_batch = batchify(X, y, batch_size, N)
    num_batches = len(y_batch)
    print("%d batches of size %d:" % (num_batches, batch_size))

    theta = theta0
    m_t = np.zeros(theta.shape)

    trace = {}
    trace['loss'] = []
    trace['theta'] = []

    for epoch in range(1, max_epoch + 1):
        loss_epoch = 0
        for b in range(num_batches):
            X_b = X_batch[b]
            y_b = y_batch[b]
            loss, gradient = loss_func(theta, X_b, y_b, *args)
            loss_epoch += loss
            theta = theta - alpha * gradient
            if np.linalg.norm(gradient) < epsilon:
                print("Gradient Descent has converged after {} epochs".format(epoch))
                break
        trace['loss'].append(np.mean(loss_epoch))
        trace['theta'].append(theta)
        if np.linalg.norm(gradient) < epsilon:
            break

    return theta, trace

def cubic_transformation(X):
    n = X.shape[1]
    phi_X = X
    phi_X = np.column_stack((phi_X, X[:, 1] * X[:, 1], X[:, 1] * X[:, 2], X[:, 2] * X[:, 2],
                        X[:, 1] * X[:, 1] * X[:, 1], X[:, 1] * X[:, 1] * X[:, 2], X[:, 1] * X[:, 2] * X[:, 2],
                        X[:, 2] * X[:, 2] * X[:, 2]))
    return phi_X

def lin_reg_loss(theta, X, y, sigma2=1):
```

```
    B = X.shape[0]
    predictions = X.dot(theta)
    error = predictions - y
    loss_f = (1 / (2 * sigma2)) * np.sum(error ** 2)
    g = (1 / (B * sigma2)) * X.T.dot(error)
    return loss_f, g

def MAP_gamma(X, y, gamma, sigma2=1):
    reg_term = gamma * sigma2 * np.identity(X.shape[1])
    theta = np.linalg.inv(X.T.dot(X) + reg_term).dot(X.T.dot(y))
    return theta.reshape(-1, 1)

def mean_square_err(X, y, theta):
    y_predict = X.dot(theta).flatten()
    mse = np.mean((y - y_predict) ** 2)
    return mse

opts = {}
opts['max_epoch'] = 100
opts['alpha'] = 1e-6
opts['tolerance'] = 1e-3
opts['batch_size'] = 10

def main():
    mu = np.zeros(10)
    sigma2 = 1
    sigma = np.identity(10) * sigma2
    mu = 0
    sigma = 1

    Ntrain = 100
    Nvalidate = 1000
    xTrain, yTrain, xValidate, yValidate = MLMAP()
    print("xTrain, yTrain, xValidate, yValidate", xTrain.shape, yTrain.shape, xValidate.shape, yValidate.shape)
    xTrain, yTrain, xValidate, yValidate = xTrain.transpose(), yTrain.flatten(), xValidate.transpose(),
yValidate.flatten()
    print("xTrain, yTrain, xValidate, yValidate", xTrain.shape, yTrain.shape, xValidate.shape, yValidate.shape)

    noiseT = multivariate_normal.rvs(mu, sigma, Ntrain)
    noiseV = multivariate_normal.rvs(mu, sigma, Nvalidate)

    xAugT = np.column_stack((np.ones(Ntrain), xTrain))
    X3train = cubic_transformation(xAugT)

    xAugV = np.column_stack((np.ones(Nvalidate), xValidate))
```

```python
X3validate = cubic_transformation(xAugV)

nCubic = X3train.shape[1]
theta0 = np.random.randn(nCubic)

theta_gd, trace = gradient_descent(lin_reg_loss, theta0, X3train, yTrain, Ntrain, **opts)
theta_MAP = MAP_gamma(X3train, yTrain, 0)

print('theta start:')
print(theta0)
print('theta MLE:')
print(theta_gd)
print('theta MAP:')
print(theta_MAP)
print()

mse_gd = mean_square_err(X3validate, yValidate, theta_gd)
mse_MAP = mean_square_err(X3validate, yValidate, theta_MAP)
print('MSE GD:', mse_gd)
print('MSE MAP:', mse_MAP)

y_MAP = X3validate.dot(theta_MAP).flatten() + noiseV
y_gd = X3validate.dot(theta_gd) + noiseV

fig_ml = plt.figure(figsize=(12, 8))
ax_ml = fig_ml.add_subplot(111, projection='3d')
ax_ml.scatter(xValidate[:, 0], xValidate[:, 1], yValidate, marker='o', color='g', label='True Data')
ax_ml.scatter(X3validate[:, 1], X3validate[:, 2], y_gd, marker='o', color='r', label='ML Estimate')
ax_ml.set_xlabel(r"$x_1$")
ax_ml.set_ylabel(r"$x_2$")
ax_ml.set_zlabel(r"$y$")
ax_ml.legend()
plt.title('True Data vs. ML Estimate')
plt.show()

fig_map = plt.figure(figsize=(12, 8))
ax_map = fig_map.add_subplot(111, projection='3d')
ax_map.scatter(xValidate[:, 0], xValidate[:, 1], yValidate, marker='o', color='g', label='True Data')
ax_map.scatter(X3validate[:, 1], X3validate[:, 2], y_MAP, marker='o', color='purple', label='MAP Estimate')
ax_map.set_xlabel(r"$x_1$")
ax_map.set_ylabel(r"$x_2$")
ax_map.set_zlabel(r"$y$")
ax_map.legend()
plt.title('True Data vs. MAP Estimate')
plt.show()
```

```python
    trials = 100000
    gamma = np.linspace(0.0000000001, 10000, trials)
    mse_range = []
    mse_range_ml = []
    theta_ml, trace_ml = gradient_descent(lin_reg_loss, theta0, X3train, yTrain, Ntrain, **opts)

    for i in range(trials):
        theta_temp = MAP_gamma(X3train, yTrain, gamma[i])
        mse_range.append(mean_square_err(X3validate, yValidate, theta_temp))
        mse_range_ml.append(mean_square_err(X3validate, yValidate, theta_ml))

    fig1 = plt.figure()
    plt.plot(gamma, mse_range, label='MAP MSE')
    plt.title("MSE vs gamma")
    plt.xlabel('gamma')
    plt.ylabel('MSE')
    plt.legend()
    plt.show()

    fig2 = plt.figure()
    plt.plot(gamma, mse_range_ml, label='ML MSE')
    plt.title("MSE vs gamma")
    plt.xlabel('gamma')
    plt.ylabel('MSE')
    plt.legend()
    plt.show()

    fig3 = plt.figure()
    plt.plot(gamma, mse_range_ml, label='ML MSE')
    plt.plot(gamma, mse_range, label='MAP MSE')
    plt.title("MSE vs gamma")
    plt.xlabel('gamma')
    plt.ylabel('MSE')
    plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

Question 3:

```
clear all;
close all;

rng('default'); % Set random number generator to default for reproducibility

% Define standard deviations for the prior distribution
sigmax = 0.25;
sigmay = 0.25;

% Create a single figure for all subplots
figure;

% Iterate over different values of K
for K = 1:4
  % Set the true vehicle location inside a circle with unit radius centered at the origin
  radius = rand;
  theta = 2 * pi * rand;
  pTrue = [radius * cos(theta); radius * sin(theta)];

  % Landmark positions evenly spaced on a circle with unit radius centered at the origin
  radius = 1;
  theta = linspace(0, 2 * pi * (1 - 1 / K), K);
  pLandmarks = [radius * cos(theta); radius * sin(theta)];

  % Set measurement noise standard deviation
  sigma = 0.3 * ones(1, K);

  % Generate range measurements according to the specified model
  r = zeros(1, K);
  while any(r < 0)
    % Reject measurements if any are negative and resample
    r = sqrt(sum((repmat(pTrue, 1, K) - pLandmarks).^2, 1)) + sigma .* randn(1, K);
  end

  % Evaluate the MAP estimation objective function on a grid
  Nx = 101;
  Ny = 99;
  xGrid = linspace(-2, 2, Nx);
  yGrid = linspace(-2, 2, Ny);
  [h, v] = meshgrid(xGrid, yGrid);

  % Compute the MAP objective
  MAPobjective = ((h - pTrue(1)) / sigmax).^2 + ((v - pTrue(2)) / sigmay).^2;
  for i = 1:K
    di = sqrt((h - pLandmarks(1, i)).^2 + (v - pLandmarks(2, i)).^2);
    MAPobjective = MAPobjective + ((r(i) - di) / sigma(i)).^2;
  end

  % Remove terms that do not impact the MAP estimate
  MAPobjective = MAPobjective - min(MAPobjective(:));
```

```
    % Create subplots
    subplot(2, 2, K);

    % Plot the equilevel contours of the MAP estimation objective
    contour(xGrid, yGrid, MAPobjective, 'ShowText', 'on');
    hold on;

    % Superimpose true location of the vehicle
    plot(pTrue(1), pTrue(2), '+r', 'MarkerSize', 10, 'LineWidth', 2);

    % Superimpose landmark locations
    plot(pLandmarks(1, :), pLandmarks(2, :), 'ob', 'MarkerSize', 8, 'LineWidth', 1);

    xlabel('x');
    ylabel('y');
    title(['MAP Objective Contours for K = ' num2str(K)]);
    grid on;
    axis equal;
end
```


Citations:1. Credits to Prof. Erdogmus Deniz

2. https://github.com/